Wiener Index, Diameter, and Stretch Factor of a Weighted Planar Graph in Subquadratic Time

Wulff-Nilsen, Christian

Publication date:
2008

Document version
Publisher's PDF, also known as Version of record

Datalogisk Institut
KØBENHAVNS UNIVERSITET

Wiener Index, Diameter, and Stretch Factor of a Weighted Planar Graph in Subquadratic Time

Christian Wulff - Nilsen

Technical Report no. 08-16
ISSN: 0107-8283

Dept. of Computer Science
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Wiener Index, Diameter, and Stretch Factor of a Weighted Planar Graph in Subquadratic Time

Christian Wulff-Nilsen *

November 12, 2008

Abstract

We solve three open problems: the existence of subquadratic time algorithms for computing the Wiener index (sum of APSP distances) and the diameter (maximum distance between any vertex pair) of a planar graph with non-negative edge weights and the stretch factor of a plane geometric graph (maximum over all pairs of distinct vertices of the ratio between the graph distance and the Euclidean distance between the two vertices). More specifically, we show that the Wiener index and diameter can be found in $O(n^2(\log\log n)^4/\log n)$ worst-case time and that the stretch factor can be found in $O(n^2(\log\log n)^4/\log n)$ expected time, where n is the number of vertices. We also show how to compute the Wiener index and diameter of an unweighted n-vertex subgraph-closed $\sqrt{n}$-separable graph in $O(n^2\log\log n/\log n)$ worst-case time and with $O(n)$ space.

1 Introduction

Computing all-pairs shortest paths (APSP) in a graph is a problem of both theoretical and practical interest and it has consequently received a lot of attention from the research community. Although solvable by repeated applications of a single-source shortest path (SSSP) algorithm, the APSP problem can be solved more efficiently for many types of graphs.

*Department of Computer Science, University of Copenhagen, koolooz@diku.dk, http://www.diku.dk/~koolooz/

The Floyd-Warshall algorithm and Johnson's algorithm are perhaps the most well-known APSP algorithms. The former assumes absence of negatively weighted cycles and has $\Theta(n^3)$ running time whereas the latter has $O(n^2 \log n + mn)$ running time, where m resp. n is the number of edges resp. vertices in the graph [6]. The fastest known APSP algorithm for dense graphs has $O(n^3(\log\log n/\log n)^{5/4})$ running time [8]. For planar graphs with non-negative edge weights, Frederickson [7] gave an optimal $\Theta(n^2)$ time algorithm.

Given a graph with non-negative edge weights, suppose we are interested in finding the *sum* of distances between all vertex pairs (or equivalently, the average distance) in the graph. This quantity is also known as the (weighted) Wiener index of the graph. Can we obtain the Wiener index in time less than what it takes to compute APSP distances? For simple graphs such as cactii, benzenoid systems, and graphs of bounded treewidth, the answer is yes [3, 4, 12, 15]. One of the main open problems in this context concerns the existence of an $o(n^2)$ time algorithm for planar graphs [3].

Another open problem (Problem 6.2 in [5]) for planar graphs is that of finding the diameter, i.e. the maximum distance between any two vertices, in subquadratic time.

The stretch factor of a geometric graph is the maximum over all pairs of distinct vertices of the ratio between the graph distance and the Euclidean distance between the two vertices. Like the two problems above, the stretch factor problem can be solved in $\Theta(n^2)$ time for plane graphs using Frederickson's APSP algorithm but it is open whether there exists a subquadratic time algorithm [1].

In this paper, we solve all three of the open problems mentioned above. More precisely, we show that the Wiener index and the diameter of an n-vertex planar graph with non-negative edge-weights can be computed in $O(n^2(\log\log n)^4/\log n)$ worst-case time and that the stretch factor of an n-vertex plane geometric graph can be computed in $O(n^2(\log\log n)^4/\log n)$ expected time.

All three results follow from relatively simple modifications of the same generic algorithm. It is our hope that this algorithm can be adapted to solve other planar graph problems more efficiently.

We also generalize the result for unweighted graphs we obtained in [14]. More specifically, we show that the Wiener index and diameter of an unweighted n-vertex subgraph-closed $\sqrt{n}$-separable graph can be found in time $O(n^2 \log\log n/\log n)$ with $O(n)$ space.

The organization of the paper is as follows. In Section 2, we introduce some notation and give some basic definitions and results. In Section 3, we describe our generic algorithm. The description of the algorithm is split into two subsections. In Subsection 3.1, we describe the preprocessing step and in Subsection 3.2, we describe the main algorithm. We also bound the running time of the entire algorithm. In Section 4, we apply the generic algorithm to the three graph problems described above and show how this gives us subquadratic time algorithms for these problems. In Section 5, we show our results for unweighted graphs. Finally, we make some concluding remarks in Section 6.

2 Definitions, Notation, and Basic Results

In this section, we introduce some notation and definitions, some of which are similar or identical to those in [2]. We also give some basic results which will prove useful.

For two vertices u and v in a graph G with non-negative edge-weights, we let $d_G(u, v)$ denote the length of a shortest path in G between u and v. If no such path exists, we define $d_G(u, v) = \infty$.

Let $G = (V, E)$ be a triangulated planar graph with a planar embedding. For a subset S of the plane, the *restriction* of G to S consists of the vertices and edges of (the embedding of) G that are fully contained in S.

When we refer to a cycle we implicitly assume that it is simple and contained in G. A cycle C partitions the plane into a bounded region, called the *interior* of C, and an unbounded region, called the *exterior* of C. We let $Int(C)$ resp. $Ext(C)$ denote the restriction of G to the interior resp. exterior of C, and we let $\overline{Int}(C)$ resp. $\overline{Ext}(C)$ denote the restriction of G to the closure of the interior resp. exterior of C.

Two cycles C and C' *do not cross* if C is contained in $\overline{Int}(C')$ or in $\overline{Ext}(C')$.

Given a collection $\mathcal{C} = \{C_1, \ldots, C_p\}$ of cycles that pairwise do not cross and where C_1 is the outer face of G, define, for $i = 1, \ldots, p$, *region* R_i by

$$R_i = \overline{Int}(C_i) - \left(\bigcup_{j \neq i, C_j \subset \overline{Int}(C_i)} Int(C_j) \right), \tag{1}$$

see Figure 1. We refer to $R_1, \ldots, R_p$ as the *regions of* G (induced by $\mathcal{C}$).

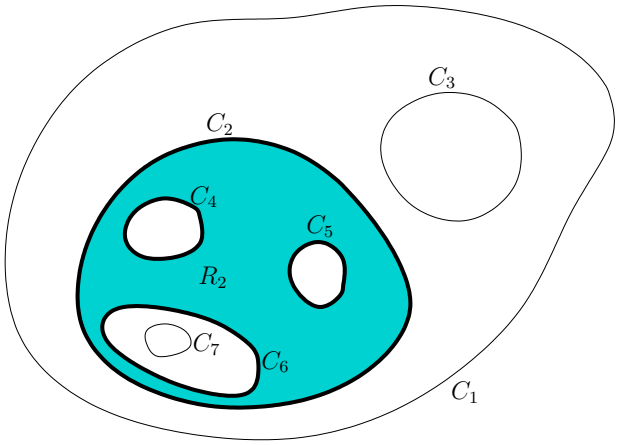

Figure 1: Example with seven cycles $C_1, \ldots, C_7$ and region R_2 (coloured) and its boundary vertices (contained in the bold curves). Also, $\mathcal{C}_{R_2} = \{C_2, C_4, C_5, C_6\}$.

For $i = 1, \ldots, p$, vertices of R_i that are adjacent to vertices in $G - R_i$ are *boundary vertices* of R_i and ∂R_i denotes the set of boundary vertices of R_i. We have the following result (taken from [2]).

Lemma 1. *Let $G = (V, E)$ be defined as above, let n be the number of vertices in G, and let $r \in (0, n)$. We can find in $O(n \log n)$ time a collection of cycles $\mathcal{C} = \{C_1, \ldots, C_p\}$ with $p = O(n/r)$ and C_1 the outer face of G such that no two cycles cross. Each of the regions $R_i \in \{R_1, \ldots, R_p\}$ of G induced by $\mathcal{C}$ has at most r vertices and ∂R_i has $O(\sqrt{r})$ vertices.*

We shall refer to the regions of G obtained by applying Lemma 1 as an *r-division* (of G).

For each region R_i, define

$$\mathcal{C}_{R_i} = \{C_j \in \mathcal{C} | C_j \subseteq \overline{Int}(C_i) \text{ and } C_j \nsubseteq \overline{Int}(C_k) \subseteq \overline{Int}(C_i) \text{ for any } k \neq i, j\},$$

see Figure 1. We observe that ∂R_i is contained in the set of vertices of the cycles of $\mathcal{C}_{R_i}$.

Given a region R_i and a cycle $C_j \in \mathcal{C}_{R_i}$, define

$$U_{R_i, C_j} = \begin{cases} Int(C_j) & \text{if } i \neq j \\ Ext(C_j) & \text{if } i = j \end{cases}$$

Lemma 2. *With the above definitions, let $i, j \in \{1, \ldots, p\}$. For any vertex $u \in U_{R_i,C_j}$ and any vertex $v \in R_i$, every shortest path in G from u to v contains a boundary vertex of R_i belonging to C_j. Furthermore, the vertices of $\cup_{C \in \mathcal{C}_{R_i}} U_{R_i,C}$ are exactly the vertices of V not belonging to R_i.*

Proof. If $i \neq j$ then $u \in \mathit{Int}(C_j)$ and $v \in R_i \subseteq \overline{\mathit{Ext}}(C_j)$. If $i = j$ then $u \in \mathit{Ext}(C_j)$ and $v \in R_i \subseteq \overline{\mathit{Int}}(C_j)$. In both cases, the first part of the lemma holds. The second part follows easily from the definitions above. □

Given two paths P_1 and P_2 in a graph, if they share an end vertex we let $P_1 \cup P_2$ denote the path obtained by concatenating P_1 and P_2.

Suppose P_1 is a shortest path between a vertex u and a vertex v_1 and that P_2 is a shortest path between u and a vertex v_2 in a graph G. Then P_1 and P_2 may be chosen so that when they split when walking from u they do not meet again. In this case, we say that P_1 and P_2 are *crossing-free* (in G). We also say that $P_1 \cup P_2$ is crossing-free.

3 A Generic Algorithm

The algorithms in Section 4 are all variations of the same generic algorithm. In this section, we describe this algorithm, not only to shorten the length of the paper but also because we believe that it can be applied to other planar graph problems involving, in some way, distances between vertex pairs.

In this section, $G = (V, E)$ denotes an n-vertex planar graph. We assume that G is triangulated (if not, we triangulate it by adding infinite weight-edges) and we compute a planar embedding of G. From this we obtain an r-division of G for some parameter r which we leave unspecified for now.

Let us start by describing the problem that the generic algorithm should solve. In the following, let R be one of the regions in the r-division of G. Let C be one of the cycles in $\mathcal{C}_R$ and let $p_1, \ldots, p_t$ be the boundary vertices of R belonging to C. Assume that in a simple walk of C, $p_1, \ldots, p_t$ are visited in that order. Let $U = U_{R,C}$.

For each $C' \in \mathcal{C}_R \setminus \{C\}$, add to R an edge between each pair of boundary vertices of R belonging to C'. The weight of this edge is equal to the length of a shortest path between the two vertices where all interior vertices on this path belong to $U_{R,C'}$. If no such path exists we omit the edge.

We refer to these extra edges as *abstract edges* since we do not specify any embedding of them. With the abstract edges added to R, note that R is

connected and still has $O(r)$ edges. Furthermore, for any vertex $u \in U$ and any vertex $v \in R$, $d_G(u,v) = d_G(u,p_i) + d_R(p_i,v)$ for some $p_i \in C$.

Now, the problem is to find, for each vertex $u \in U$, a colouring of the vertices of R using t colours which we denote by indices $1,\ldots,t$. For $i = 1,\ldots,t$, a vertex $v \in R$ should be assigned colour i if $d_G(u,v) = d_G(u,p_i) + d_R(p_i,v)$. Ties may be resolved in any way. We refer to such a colouring as a *u-colouring* (of R).

By the above, it follows that a u-colouring always exists. Furthermore, if a vertex $v \in R$ is given colour i in a u-colouring then there is a shortest path in G from u to v through p_i with the subpath from p_i to v contained in R. As we shall see, this information enables us to efficiently solve our three problems of Section 4, given an efficient algorithm that finds u-colourings.

For performance reasons, a u-colouring should not be computed explicitly. Instead, a key value should be efficiently calculated that uniquely identifies that colouring. The idea is then to use this key as an index into a table in which either precomputations have been made (in case of the Wiener index and the diameter problem) or where vertices inducing identical colourings can be grouped together to speed up subsequent computations (in case of the stretch factor problem).

3.1 Preprocessing Step

The main algorithm is preceeded by a preprocessing step. In this subsection, we describe this step. In the following, assume that R, C, U, and $p_1,\ldots,p_t$ are defined as above. We make the assumption that C is a cycle such that R is contained in $\overline{Ext}(C)$. The case where C is the cycle such that R is contained in $\overline{Int}(C)$ is dealt with in a similar way.

The main algorithm needs to find a u-colouring of R w.r.t. each vertex $u \in U$. The preprocessing step does not depend on u. However, to make it more clear how the preprocessing can speed up computations in the main algorithm, we assume in the following that we are given some unspecified $u \in U$.

When convenient, we regard an abstract edge between two boundary vertices of R belonging to a cycle C' as a shortest path between the vertices having all its interior vertices in $U_{R,C'}$. By definition, the weight of the abstract edge is equal to the length of this path. The following lemma will prove useful as it allows us to obtain information about such paths without explicitly knowing them (see Figure 2).

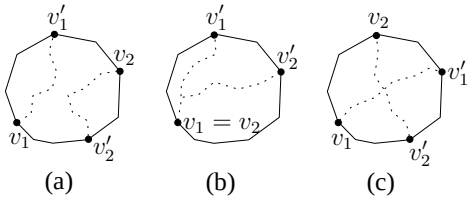

Figure 2: In (a) and (b), paths represented by abstract edges $e_1 = (v_1, v_1')$ and $e_2 = (v_2, v_2')$ can be chosen such that they do not cross. In (c) they have to cross.

Lemma 3. *Let $C' \in \mathcal{C}_R \setminus \{C\}$ and let $e_1 = (v_1, v_1')$ and $e_2 = (v_2, v_2')$ be abstract edges, where $v_1, v_1', v_2, v_2' \in C'$. Then the paths represented by e_1 and e_2 may be chosen such that they do not cross if and only if there is a cyclic walk of C' that visits v_1, v_1', v_2, and v_2' in one of the following two orders:*

1. *v_1, v_1', v_2, v_2' (where possibly $v_1' = v_2$ or $v_1 = v_2'$),*
2. *v_1, v_1', v_2', v_2 (where possibly $v_1' = v_2'$ or $v_1 = v_2$).*

Proof. This is an easy consequence of planarity and the fact that the interior vertices of the shortest paths represented by e_1 and e_2 are all contained in $\overline{Int}(C')$ or all contained in $\overline{Ext}(C')$. □

Let $\mathcal{I}_t = \{1, \ldots, t\}$. For non-empty subset $\mathcal{I} \subseteq \mathcal{I}_t$, define an *$\mathcal{I}$-colouring of R* as a colouring of the vertices of R with colours $i \in \mathcal{I}$. We extend this definition to subgraphs of R and to subsets of vertices of R. When less specific, we call an $\mathcal{I}$-colouring of R an $|\mathcal{I}|$-colouring of R. When convenient, we will regard an $\mathcal{I}$-colouring of a set A of vertices as a map $c : A \to \mathcal{I}$.

An *$\mathcal{I}$-colouring of R w.r.t. u* is an $\mathcal{I}$-colouring of R where a vertex $v \in R$ is given colour i only if $d_G(u, p_i) + d_R(p_i, v) \leq d_G(u, p_j) + d_R(p_j, v)$ for all $j \in \mathcal{I}$, with ties resolved in any way. Given a subset A of vertices of R, an $\mathcal{I}$-colouring of A w.r.t. u is defined as an $\mathcal{I}$-colouring of A which can be extended to an $\mathcal{I}$-colouring of R w.r.t. u.

For an index set $\mathcal{I} = \{i_1, \ldots, i_m\}$ with $i_1 < i_2 < \ldots < i_m$, define the *lower subset* of $\mathcal{I}$ as the set $\{i_1, \ldots, i_{\lceil m/2 \rceil}\}$ and define the *upper subset* of $\mathcal{I}$ as the set $\{i_{\lceil m/2 \rceil + 1}, \ldots, i_m\}$.

The generic algorithm should find an $\mathcal{I}_t$-colouring of R w.r.t. u. The idea is to obtain this colouring recursively from an $\mathcal{I}_1$-colouring and an $\mathcal{I}_2$-colouring of R w.r.t. u, where $\mathcal{I}_1$ ($\mathcal{I}_2$) is the lower (upper) subset of $\mathcal{I}_t$.

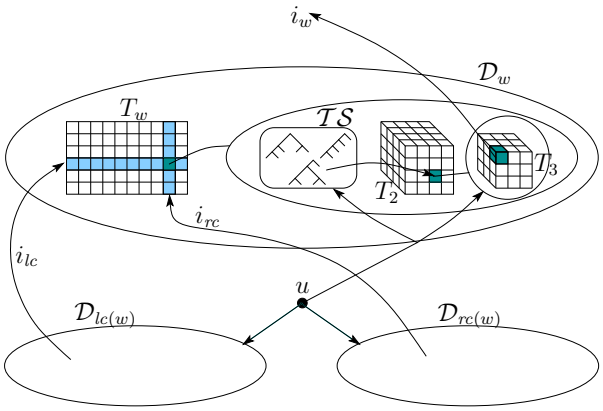

Figure 3: Data structures $\mathcal{D}_{lc(w)}$ and $\mathcal{D}_{rc(w)}$ are recursively queried with u, giving two numbers i_{lc} and i_{rc}, respectively. From these the tree set $\mathcal{TS}$ and the level 2-table T_2 associated with entry (i_{lc}, i_{rc}) in level 1-table T_w are obtained. Querying $\mathcal{TS}$ with u gives a binary tree which is mapped to an entry in T_2. Associated with this entry is a level 3-table T_3. Querying with u gives an entry in T_3 which has an associated number i_w defining an $\mathcal{I}_w$-colouring of R w.r.t. u.

In the preprocessing step of the algorithm, a balanced binary tree $\mathcal{T}$ is constructed that reflects this recursion. We refer to this tree as the *main tree*. Associated with each vertex w of $\mathcal{T}$ is a subset $\mathcal{I}_w$ of $\mathcal{I}_t$.

Main tree $\mathcal{T}$ and these subsets are defined as follows. The root r of $\mathcal{T}$ is associated with $\mathcal{I}_r = \mathcal{I}_t$. For each vertex w of $\mathcal{T}$, if $|\mathcal{I}_w| = 1$ then w is a leaf of $\mathcal{T}$. Otherwise, w has a left child which is associated with the lower subset of $\mathcal{I}_w$ and has a right child which is associated with the upper subset of $\mathcal{I}_w$. We let $\mathcal{T}_w$ denote the subtree of $\mathcal{T}$ rooted at w.

For each non-leaf vertex w of $\mathcal{T}$, we let $lc(w)$ resp. $rc(w)$ denote the left resp. right child of w. With each such w we associate a data structure, called $\mathcal{D}_w$. When queried with u in the main algorithm, $\mathcal{D}_w$ returns an integer that identifies an $\mathcal{I}_w$-colouring of R w.r.t. u. We start by giving an overview of this data structure. Below, we describe it in more detail.

Data structure $\mathcal{D}_w$ is illustrated in Figure 3. It consists of a two-dimensional table T_w which we call the *level* 1*-table* (of w). Associated with each entry $T_w(i, j)$ of T_w is a data structure called a *tree set*. It represents a certain set of binary trees and there is a one-to-one map from this set into a higher-dimensional table, also associated with entry $T_w(i, j)$. We call this table a

level 2*-table*. With each entry of this table, a higher-dimensional table called a *level* 3*-table* is associated.

Each level 3-table entry associated with $\mathcal{D}_w$ corresponds to an $\mathcal{I}_w$-colouring of R. When $\mathcal{D}_w$ is queried with u in the main algorithm, this vertex will be mapped to a level 3-table entry that corresponds to an $\mathcal{I}_w$-colouring of R w.r.t. u. A unique number is assigned to each level 3-table entry and if u is mapped to an entry with number x then x is the key value that uniquely identifies that $\mathcal{I}_w$-colouring of R w.r.t. u. It is this key value that $\mathcal{D}_w$ returns when queried with u.

In the preprocessing step, the actual colourings associated with level 3-entries will be needed. These colourings will be computed bottom-up in main tree $\mathcal{T}$.

In the following, we consider a non-leaf vertex w of $\mathcal{T}$ and describe the components of data structure $\mathcal{D}_w$ associated with w. We assume that colourings of level 3-table entries associated with descendants of w in $\mathcal{T}$ have already been computed.

3.1.1 Level 1-Table

The index of each row of level 1-table T_w is an integer representing an $\mathcal{I}_{lc(w)}$-colouring of R w.r.t. some vertex and the index of each column is an integer representing an $\mathcal{I}_{rc(w)}$-colouring of R w.r.t. some vertex. There is a row resp. column for each level 3-table entry associated with $\mathcal{D}_{lc(w)}$ resp. $\mathcal{D}_{rc(w)}$ and each entry of T_w is associated with the two colourings corresponding to the entry's row and column, respectively.

In the main algorithm, data structures $\mathcal{D}_{lc(w)}$ and $\mathcal{D}_{rc(w)}$ are recursively queried with vertex u. This gives two integers i_{lc} and i_{rc} that identify, respectively, an $\mathcal{I}_{lc(w)}$-colouring and an $\mathcal{I}_{rc(w)}$-colouring of R w.r.t. u. These two integers are then used as indices to obtain entry (i_{lc}, i_{rc}) in T_w, see Figure 3.

In case $lc(w)$ is a leaf of $\mathcal{T}$ then there is only one $\mathcal{I}_{lc(w)}$-colouring of R w.r.t. u so we define $i_{lc} = 1$ and T_w has only one row. And similarly, if $rc(w)$ is a leaf of $\mathcal{T}$ then $i_{rc} = 1$ and T_w has only one column.

In the following, we describe the tree set and the level 2-table and level 3-tables associated with the entry (i_{lc}, i_{rc}) of T_w that u is mapped to in the main algorithm.

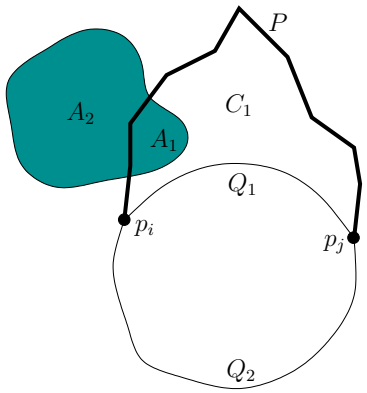

Figure 4: Crossing-free path P partitions C into two subpaths, Q_1 and Q_2, and partitions A (coloured region) into two subsets, A_1 and A_2.

3.1.2 Tree Set

The tree set data structure is essentially a compact representation of a certain set of binary trees and has a recursive definition. It is either a *leaf* or a *non-leaf*. If it is a leaf it represents one binary tree which itself is a leaf.

If it is a non-leaf it consists of a pair of arrays and each entry of these arrays represents a root of a set of binary trees. An entry corresponding to a root r points to two recursively defined tree sets representing, respectively, left and right subtrees of a binary tree with root r. Taking all combinations of left and right subtrees that those two tree sets represent gives all binary trees with root r.

Before describing the tree set in greater detail, we need some more definitions and results.

Consider a crossing-free path P in R (where each abstract edge is regarded as a path) between two boundary vertices p_i and p_j of R belonging to C. The two boundary vertices partition C into two subpaths, Q_1 and Q_2 (both containing p_i and p_j), see Figure 4. Observe that $C_1 = Q_1 \cup P$ and $C_2 = Q_2 \cup P$ are cycles and that either C_1 is contained in $\overline{Int}(C_2)$ or C_2 is contained in $\overline{Int}(C_1)$. Assume w.l.o.g. that C_1 is contained in $\overline{Int}(C_2)$.

Let A be a subset of vertices of R. Define A_1 resp. A_2 as the subset of vertices of A belonging to $\overline{Int}(C_1)$ resp. $\overline{Ext}(C_1)$. We refer to A_1 and A_2 as the *subsets of A induced by P*. For $m = 1, 2$, we say that the set of indices of boundary vertices in Q_m are *associated* with A_m.

Now, assume we are given a subset A of vertices of R, two subsets $\mathcal{I} \subseteq$

$\mathcal{I}_{lc(w)}$ and $\mathcal{I}' \subseteq \mathcal{I}_{rc(w)}$, an $\mathcal{I}$-colouring c of A w.r.t. u, and an $\mathcal{I}'$-colouring c' of A w.r.t. u. For a vertex $v \in A$, we refer to $(c(v), c'(v))$ as the *colour pair* of v (w.r.t. c and c').

Let v be a vertex in A with colour pair (i, j). We call v a *split vertex of* A (w.r.t. u and colourings c and c') if there exists a shortest path P_i resp. P_j in R from p_i resp. p_j to v with all its vertices in A, if P_i and P_j are crossing-free, and if

1. $d_G(u, p_i) + d_R(p_i, v) \leq d_G(u, p_j) + d_R(p_j, v)$ and
2. $d_G(u, p_j) + d_R(p_j, v') < d_G(u, p_k) + d_R(p_k, v')$ for any vertex $v' \in P_j \setminus \{v\}$ and any $k \in \mathcal{I}$,

or if the same two conditions hold with i and j swapped and $\mathcal{I}$ replaced by $\mathcal{I}'$. In this case, we refer to P_i and P_j as the *split paths* associated with v.

As we shall see, the following lemma will help us "merge" the given $\mathcal{I}_{lc(w)}$-colouring and the given $\mathcal{I}_{rc(w)}$-colouring into an $\mathcal{I}_w$-colouring of R w.r.t. u.

Lemma 4. *With the above definitions, suppose that v is a split vertex of A with colour pair (i, j) w.r.t. colourings c and c' and let P_i and P_j be split paths associated with v. Let A_1 and A_2 be the subsets of A induced by $P_i \cup P_j$. For $m = 1, 2$, let $\mathcal{I}_m$ be the index set associated with A_m. Then there is an $\mathcal{I} \cup \mathcal{I}'$-colouring of A_m w.r.t. u where each colour $k \in \mathcal{I}_m$.*

Proof. By symmetry, it suffices to show the lemma for $m = 1$. So let $a \in A_1$ be given. We will show that in an $\mathcal{I} \cup \mathcal{I}'$-colouring of A_1 w.r.t. u, a can be assigned a colour $k \in \mathcal{I}_1$.

Since v is a split vertex of A, we may assume, again by symmetry, that $d_G(u, p_i) + d_R(p_i, v) \leq d_G(u, p_j) + d_R(p_j, v)$ and that $d_G(u, p_j) + d_R(p_j, v') < d_G(u, p_k) + d_R(p_k, v')$ for any vertex $v' \in P_j \setminus \{v\}$ and any $k \in \mathcal{I}$.

Pick $k \in \mathcal{I} \cup \mathcal{I}'$ such that $d_G(u, p_k) + d_R(p_k, a) \leq d_G(u, p_{k'}) + d_R(p_{k'}, a)$ for all $k' \in \mathcal{I} \cup \mathcal{I}'$. Suppose that $k \notin \mathcal{I}_1$. Then a shortest path in R from p_k to a must contain some vertex $v' \in P_i \cup P_j$. Assume first that $v' \in P_i$, see Figure 5(a). Since $d_G(u, p_i) + d_R(p_i, v) \leq d_G(u, p_j) + d_R(p_j, v)$, any vertex on P_i, and in particular v', can be assigned colour i in an $\mathcal{I} \cup \mathcal{I}'$-colouring of A_1 w.r.t. u. This implies that a can be assigned colour $i \in \mathcal{I}_1$ in such a colouring, as requested.

Now, suppose $v' \in P_j \setminus \{v\}$, see Figure 5(b). Either $k \in \mathcal{I}$ or $k \in \mathcal{I}'$. If $k \in \mathcal{I}'$ then we may assign colour $j \in \mathcal{I}_1$ to a in an $\mathcal{I} \cup \mathcal{I}'$-colouring so assume that $k \in \mathcal{I}$. Then $d_G(u, p_j) + d_R(p_j, v') < d_G(u, p_k) + d_R(p_k, v')$ by the

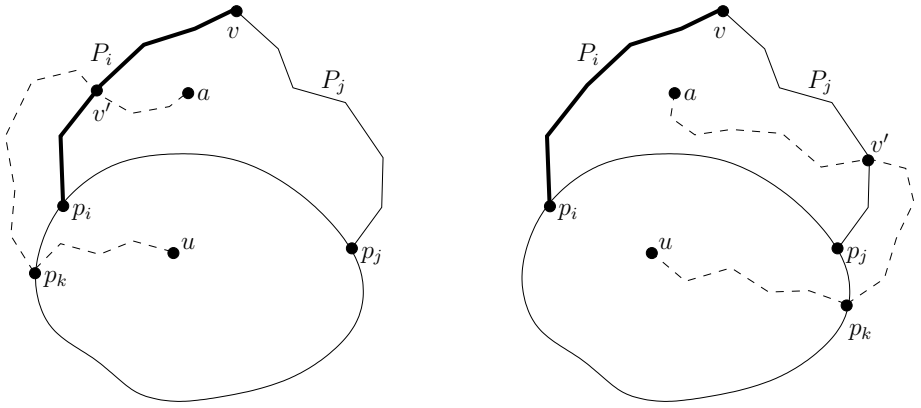

Figure 5: A shortest path in R from p_k to a intersects either P_i or $P_j \setminus \{v\}$ in some vertex v'.

assumption that v is a split vertex of A. By the choice of k and the triangle inequality,

$$\begin{aligned} d_G(u,p_k) + d_R(p_k,v') + d_R(v',a) &= d_G(u,p_k) + d_R(p_k,a) \\ &\leq d_G(u,p_j) + d_R(p_j,a) \\ &\leq d_G(u,p_j) + d_R(p_j,v') + d_R(v',a), \end{aligned}$$

implying that $d_G(u,p_k) + d_R(p_k,v') \leq d_G(u,p_j) + d_R(p_j,v')$. But this contradicts the inequality above. Hence, the lemma holds in all cases. □

Now, let us give a more detailed description of the tree set associated with an entry of level 1-table T_w and how it is constructed. Let us denote it by $\mathcal{TS}$.

We use a recursive algorithm for the construction. Its input is a tuple $(A, \mathcal{I}, \mathcal{I}', c, c')$ where A is a subset of vertices of R and c and c' are colourings of A. The following invariant will be maintained:

1. c is an $\mathcal{I}$-colouring and c' is an $\mathcal{I}'$-colouring of A w.r.t. u,
2. for any $v \in A$ there is a shortest path P in R from $p_{c(v)}$ to v, and a shortest path P' in R from $p_{c'(v)}$ to v such that all vertices on these paths belong to A, and

3. the indices of $\mathcal{I}$ resp. $\mathcal{I}'$ are consecutive in the cyclic ordering induced by C.

Let c_{lc} resp. c_{rc} be the $\mathcal{I}_{lc(w)}$- resp. $\mathcal{I}_{rc(w)}$-colouring of R w.r.t. u associated with the entry of T_w that we consider. To construct $\mathcal{TS}$, we call this algorithm with A the set of vertices of R, $\mathcal{I} = \mathcal{I}_{lc(w)}$, $\mathcal{I}' = \mathcal{I}_{rc(w)}$, $c = c_{lc}$, and $c' = c_{rc}$. Note that the invariant of the algorithm holds trivially in this case.

Now, consider an invocation with input $(A, \mathcal{I}, \mathcal{I}', c, c')$ and let us describe how a tree set $\mathcal{TS}'$ is constructed by the algorithm.

Let A' be the subset of vertices v of A for which $c(v)$ resp. $c'(v)$ is neither the first nor last index of $\mathcal{I}$ resp. $\mathcal{I}'$. If $A' = \emptyset$, a leaf is returned.

Otherwise, we pick a vertex $v \in A'$. Let (i, j) be the colour pair of v and let P_i and P_j be shortest paths in R from p_i to v and from p_j to v, respectively. By the invariant, we may pick P_i and P_j such that all their vertices belong to A and as observed in Section 2, we may pick them such that P_i and P_j are crossing-free (when regarding each abstract edge as a path). By Lemma 3, we can ensure that P_i and P_j are crossing-free without knowing the paths represented by the abstract edges.

Next, we redefine colourings c and c'. More specifically, for each vertex $v' \in P_i$, set $c(v') := i$ and for each vertex $v' \in P_j$, set $c'(v') := j$. Since (i, j) is the colour pair of v, this change does not violate the invariant of the algorithm.

In the main algorithm, we will pick a split vertex in $P_i \cup P_j$. Roughly, the idea is to apply Lemma 4 on this vertex to divide the problem of finding an $\mathcal{I} \cup \mathcal{I}'$-colouring of R w.r.t. u into two subproblems which are then recursed on.

Let $\mathcal{A}_i$ and $\mathcal{A}_j$ be the two arrays associated with the top-level of $\mathcal{TS}'$. In the preprocessing step, we do not know which will be the split vertex w.r.t. u so we need to consider every possible vertex of P_i and of P_j. We therefore define $\mathcal{A}_i$ to have an entry for each vertex of P_i and define $\mathcal{A}_j$ to have an entry for each vertex of P_j. The entries of $\mathcal{A}_i$ are ordered according to how the vertices occur when walking from p_i to v in P_i. The entries of $\mathcal{A}_j$ are ordered similarly for P_j.

The entry of $\mathcal{A}_i$ resp. $\mathcal{A}_j$ corresponding to a vertex v' is called the v'-*entry* of $\mathcal{A}_i$ resp. $\mathcal{A}_j$. We also say that it is the v'-entry of $\mathcal{TS}'$.

With each vertex $v' \in P_i$, two paths, $P_i(v')$ and $P'_i(v')$, are associated, see Figure 6. These paths will be chosen as the split paths if v' is selected as split vertex in the main algorithm. Path $P_i(v')$ is the subpath of P_i from

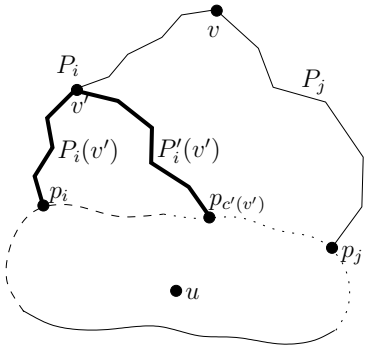

Figure 6: Associated with each vertex $v' \in P_i$ are two paths, $P_i(v')$ and $P'_i(v')$. For this example, the dashed resp. dotted curve illustrates the portion of C containing boundary vertices with indices in $\mathcal{I}$ resp. $\mathcal{I}'$.

p_i to v' and $P'_i(v')$ is a shortest path in R from $p_{c'(v')}$ to v' such that all vertices on this path belong to A (here we use the invariant again) and such that $P_i(v')$ and $P'_i(v')$ are crossing-free. Similarly, associate paths $P_j(v')$ and $P'_j(v')$ with each vertex $v' \in P_j$, where $P_j(v')$ is the subpath of P_j from p_j to v' and $P'_j(v')$ is a shortest path in R from $p_{c(v')}$ to v' such that all vertices on this path belong to A and such that $P_j(v')$ and $P'_j(v')$ are crossing-free.

Consider some v'-entry of $\mathcal{A}_i$. We now describe how the two tree sets that this entry points to are recursively constructed ($\mathcal{A}_j$ is dealt with in a similar way).

Let A_1 and A_2 be the subsets of A induced by $P_i(v') \cup P'_i(v')$ and let $j' = c'(v')$. For $m = 1, 2$, let $\mathcal{I}_m = \mathcal{J}_m \cap \mathcal{I}$ and $\mathcal{I}'_m = \mathcal{J}_m \cap \mathcal{I}'$, where $\mathcal{J}_m$ is the index set associated with A_m.

Next, we define two colourings of A_1, $c_1 : A_1 \to \mathcal{I}_1$ and $c'_1 : A_1 \to \mathcal{I}'_1$. For a vertex $v_1 \in A_1$, if $c(v_1) \in \mathcal{I}_1$ then $c_1(v_1) = c(v_1)$ and otherwise, $c_1(v_1) = i$. And if $c'(v_1) \in \mathcal{I}'_1$ then $c'_1(v_1) = c'(v_1)$ and otherwise, $c'_1(v_1) = j'$. We define colourings $c_2 : A_2 \to \mathcal{I}_2$ and $c'_2 : A_2 \to \mathcal{I}'_2$ of A_2 in a similar manner.

If we go through the proof of Lemma 4, we see that, for $m = 1, 2$, c_m resp. c'_m is an $\mathcal{I}_m$- resp. $\mathcal{I}'_m$-colouring of A_m w.r.t. u.

To construct the two tree sets that the v'-entry points to, the algorithm calls itself recursively with input tuples $(A_1, \mathcal{I}_1, \mathcal{I}'_1, c_1, c'_1)$ and $(A_2, \mathcal{I}_2, \mathcal{I}'_2, c_2, c'_2)$, respectively. From the above, it follows that the invariant of the algorithm holds for these two recursive calls.

The above is repeated for all entries of $\mathcal{A}_i$ and for all entries of $\mathcal{A}_j$ and the

algorithm returns the resulting tree set $\mathcal{TS}'$. This completes the description of the algorithm as well as the definition of $\mathcal{TS}$.

For a tree set $\mathcal{TS}'$ in the above recursive construction, we say that A, $\mathcal{I}$, $\mathcal{I}'$, picked vertex v and paths P_i and P_j are *associated* with $\mathcal{TS}'$. We also say that the colourings c and c' are associated with $\mathcal{TS}'$ after they have been redefined in the algorithm as described above. Finally, we say that the colour pairs in $\mathcal{I} \times \mathcal{I}'$ of vertices in A and the indices of $\mathcal{I} \cup \mathcal{I}'$ are associated with $\mathcal{TS}'$.

In the main algorithm, when $\mathcal{TS}$ is queried with vertex $u \in U$, a certain tree of $\mathcal{TS}$ is traversed. We refer to this traversal as a *u-traversal* of $\mathcal{TS}$ and we say that *u traverses* $\mathcal{TS}$.

For a tree set $\mathcal{TS}'$ in the above construction, a u-traversal of $\mathcal{TS}'$ is defined as follows. If $\mathcal{TS}'$ represents a leaf then this leaf constitutes the u-traversal.

Otherwise, let A, v, $\mathcal{I} \subseteq \mathcal{I}_{lc(w)}$, and $\mathcal{I}' \subseteq \mathcal{I}_{rc(w)}$ be associated with $\mathcal{TS}'$. Let (i, j) be the colour pair of v w.r.t. the colourings associated with $\mathcal{TS}'$.

If $d_G(u, p_i) + d_R(p_i, v) \leq d_G(u, p_j) + d_R(p_j, v)$ then a split vertex v' of A is picked such that it corresponds to an entry in array $\mathcal{A}_j$. Otherwise, it is picked such that it corresponds to an entry in $\mathcal{A}_i$. It follows from the definition of split vertex that v' must exist. The u-traversal of $\mathcal{TS}'$ consists of v' and recursive u-traversals of the tree sets that the v'-entry points to.

Lemma 8 below shows that $\mathcal{I}_w$-colourings of R w.r.t. vertices traversing the same binary tree are related in a sense. To prove this lemma, we need the following three results.

Lemma 5. *In the above construction,* $|\mathcal{I}_1 \cup \mathcal{I}'_1| + |\mathcal{I}_2 \cup \mathcal{I}'_2| = |\mathcal{I} \cup \mathcal{I}'| + 2$ *and* $|\mathcal{I}_1 \cup \mathcal{I}'_1|, |\mathcal{I}_2 \cup \mathcal{I}'_2| < |\mathcal{I} \cup \mathcal{I}'|$.

Proof. With j' defined as above, the first part follows from the observation that $\mathcal{I} \cup \mathcal{I}' = \mathcal{I}_1 \cup \mathcal{I}'_1 \cup \mathcal{I}_2 \cup \mathcal{I}'_2$ and that i and j' are the only indices of $\mathcal{I} \cup \mathcal{I}'$ shared by $\mathcal{I}_1 \cup \mathcal{I}'_1$ and $\mathcal{I}_2 \cup \mathcal{I}'_2$.

To show the second part, suppose vertex v' above corresponds to an entry in array $\mathcal{A}_i$. By the third part of the invariant of the algorithm, $\mathcal{I}_1$ contains an index k which is either the first or the last index of $\mathcal{I}$. Since $v \in A'$, i is neither the first nor last index of $\mathcal{I}$ so $k \neq i$. The only index of $\mathcal{I}$ shared by $\mathcal{I}_1$ and $\mathcal{I}_2$ is i so $k \notin \mathcal{I}_2$. Hence, $k \notin \mathcal{I}_2 \cup \mathcal{I}'_2$, implying that $|\mathcal{I}_2 \cup \mathcal{I}'_2| < |\mathcal{I} \cup \mathcal{I}'|$. A similar argument shows that $|\mathcal{I}_1 \cup \mathcal{I}'_1| < |\mathcal{I} \cup \mathcal{I}'|$. The inequalities also follow if v' corresponds to an entry in $\mathcal{A}_j$. □

Lemma 6. *Each binary tree represented by $\mathcal{TS}$ has $O(|\mathcal{I}_w|)$ vertices.*

Proof. Consider a tree represented by $\mathcal{TS}$. Lemma 5 implies that the number of leaves in this tree is at most $l(|\mathcal{I}_w|)$, where $l : \mathbb{N} \to \mathbb{N}$ is defined by

$$l(k) = \begin{cases} \max\{l(k_1) + l(k_2) | k_1 + k_2 = k + 2, k_1, k_2 < k\} & \text{if } k > 4 \\ 1 & \text{if } k \leq 4, \end{cases}$$

where we used the observation that $A' = \emptyset$ when $|\mathcal{I}_w| \leq 4$ in the algorithm that constructs $\mathcal{TS}$. Since the number of non-leaf nodes is one less than the number of leaves, the lemma will follow if we can show that $l(k) \leq k - 2$ for $k \geq 3$.

The proof is by induction on $k \geq 3$. If $k = 3$, we have $l(k) = 1 = k - 2$ by definition so assume that $k > 3$ and that the induction hypothesis holds for smaller values than k. Let k_1, k_2 be given where $k_1 + k_2 = k + 2$ and $k_1, k_2 < k$. Then $l(k_1) + l(k_2) \leq k_1 + k_2 - 4 = k - 2$. Hence,

$$l(k) = \max\{l(k_1) + l(k_2) | k_1 + k_2 = k + 2, k_1, k_2 < k\} \leq k - 2,$$

as requested. □

Lemma 7. *Given a binary tree T of $\mathcal{TS}$, the total number of distinct colour pairs associated with leaves of T is $O(|\mathcal{I}_w|)$.*

Proof. Let T be a tree of $\mathcal{TS}$. For any of its non-leaf vertices, if there are k indices associated with this vertex then the total number of indices associated with its two children is $k + 2$ by Lemma 5. Hence, if m is the number of non-leaf vertices of T then the total number of indices associated with leaves of T is $2m + |\mathcal{I}_w|$ which is $O(|\mathcal{I}_w|)$ by Lemma 6.

Let $\mathcal{I} \subseteq \mathcal{I}_{lc(w)}$ and $\mathcal{I}' \subseteq \mathcal{I}_{rc(w)}$ be the index sets and let A be the subset of vertices associated with a leaf of T. By the above, the lemma will follow if we can show that the number of distinct colour pairs in $\mathcal{I} \times \mathcal{I}'$ of vertices in A is $O(|\mathcal{I} \cup \mathcal{I}'|)$.

Since we are in a leaf of T, we have that for any such colour pair (i, j), either i is the first or last index of $\mathcal{I}$ or j is the first or last index of $\mathcal{I}'$. This implies that the number of distinct colour pairs is at most $2|\mathcal{I} \cup \mathcal{I}'|$, as requested. □

Lemma 8. *Let U' be a subset of vertices of U all traversing the same binary tree in $\mathcal{TS}$. Then there are $O(|\mathcal{I}_w|)$ sets $V_1, \ldots, V_m$ of vertices whose union is the set of vertices of R such that for $k = 1, \ldots, m$ and for each $u' \in U'$ there is a 2-colouring of V_k which is an $\mathcal{I}_w$-colouring of V_k w.r.t. u'.*

Proof. Let $u \in U'$. Define $A_1, \ldots, A_p$ as the subsets of vertices associated with leaves of the binary tree traversed by u. From the construction of $\mathcal{TS}$ and the definition of u-traversal, it follows that $\cup_{k=1}^{p} A_k$ is the set of vertices of R.

Let $k \in \{1, \ldots, p\}$ be given and let l_k denote the leaf with which A_k is associated. Let $c : A_k \to \mathcal{I}$ and $c' : A_k \to I'$ be the colourings of A_k associated with l_k, where $\mathcal{I} \subseteq \mathcal{I}_{lc(w)}$ and $\mathcal{I}' \subseteq \mathcal{I}_{rc(w)}$.

By the invariant of the algorithm that constructs $\mathcal{TS}$, it follows that for each $v \in A_k$, either $c(v)$ or $c'(v)$ is a colour of v in an $\mathcal{I}_w$-colouring of R w.r.t. u. The lemma follows from Lemma 7 by letting $V_1, \ldots, V_m$ be sets each containing vertices with identical colour pairs. □

3.1.3 Level 2-Table

We now describe the level 2-tables associated with $\mathcal{D}_w$. Recall that there is a tree set associated with each entry of the level 1-table T_w. For each such tree set $\mathcal{TS}$ there is a one-to-one map ϕ from the set of binary trees that $\mathcal{TS}$ represents to entries of a level 2-table. In the following, we define this map and table.

Let $c \in \mathbb{N}$ be a constant such that each tree represented by $\mathcal{TS}$ has at most $c|\mathcal{I}_w|$ non-leaf vertices. Such a constant exists by Lemma 6. Let $n_R = O(r)$ be the number of vertices of R. Arbitrarily assign a unique number in $\{0, \ldots, n_R - 1\}$ to each of these vertices.

Consider a tree T represented by $\mathcal{TS}$. Its root corresponds to an entry in one of the two arrays at the top-level of $\mathcal{TS}$. This entry is uniquely defined by the choice of array and the choice of split vertex. The array is uniquely determined by the value of a single bit and the split vertex is uniquely determined by its number in the above assignment.

It follows that the root of T is uniquely defined by a pair in $\{0, 1\} \times \{0, \ldots, n_R - 1\}$ which we may regard as a pair in $\{0, \ldots, n_R - 1\}^2$.

Applying the above recursively to the subtrees of T, it follows that T is uniquely defined by a vector in $\{0, \ldots, n_R - 1\}^{2c|\mathcal{I}_w|}$. We define $\phi(T)$ to be this vector and with this definition, ϕ is one-to-one.

The level 2-table T_2 corresponding to $\mathcal{TS}$ is the $2c|\mathcal{I}_w|$-dimensional table with entries $\{0, \ldots, n_R - 1\}^{2c|\mathcal{I}_w|}$.

3.1.4 Level 3-Table

What remains in order to complete the description of data structure $\mathcal{D}_w$ is to define the level 3-tables associated with this data structure. To understand why we need these tables, we need the following observations.

Let U' be a set of vertices of U that traverse the same tree in $\mathcal{TS}$. Then $\phi(U') = \{\overline{v}\}$ for some vector $\overline{v}$ corresponding to an entry in T_2.

Let $V_1, \ldots, V_m$ be defined as in Lemma 8. Then the same lemma implies that for each $u' \in U'$ it is enough to specify 2-colourings of $V_1, \ldots, V_m$ to specify an $\mathcal{I}_w$-colouring of R w.r.t. u'. The following lemma shows that each of these 2-colourings can be specified by an integer in $\{0, \ldots, n_R\}$ and that this integer can be computed efficiently.

Lemma 9. *Let i and j be distinct indices in $\mathcal{I}_w$. With the above definitions, there is a map $f : U' \to \{0, \ldots, n_R\}$ and $\{i, j\}$-colourings $c_0, \ldots, c_{n_R}$ of R such that for each $u \in U'$, $c_{f(u)}$ is an $\{i, j\}$-colouring of R w.r.t. u. Assuming $d_G(u, p_i)$ and $d_G(u, p_j)$ are given, $f(u)$ can be computed in $O(\log r)$ time for any $u \in U'$ with preprocessing time polynomial in r.*

Proof. Assume first that $D = d_R(p_i, p_j) < \infty$. For any $u \in U'$, the triangle inequality implies that $|d_G(u, p_i) - d_G(u, p_j)| \leq D$. Observe that if $d_G(u, p_i) - d_G(u, p_j) = -D$ then there is an $\{i, j\}$-colouring of R w.r.t. u where all vertices of R have colour c_i. And if $d_G(u, p_i) - d_G(u, p_j) = D$ then there is an $\{i, j\}$-colouring where all vertices of R have colour c_j.

Consider adding a new vertex u to G and edges $e_i = (u, p_i)$ and $e_j = (u, p_j)$. Set the weights of e_i and e_j such that $x = -D$, where $x = d_G(u, p_i) - d_G(u, p_j)$. Now, consider adjusting the weights such that x is increased continuously from $-D$ to D.

Initially, all vertices of R have colour c_i in the $\{i, j\}$-colouring of R w.r.t. u. There are event points in $[-D, D]$, where the colouring changes. Such changes occur exactly when $d_G(u, p_i) + d_R(p_i, v) = d_G(u, p_j) + d_R(p_j, v)$, i.e. when $x = d_R(p_j, v) - d_R(p_i, v)$ for some $v \in R$.

Since there are n_R vertices in R, we have shown that there are at most $n_R + 1$ distinct $\{i, j\}$-colourings and that each colouring corresponds to an interval between two consecutive event points in $[-D, D]$. By ordering the event points, we can apply binary search to find, in $O(\log r)$ time, the interval corresponding to an $\{i, j\}$-colouring of R w.r.t. a vertex $u \in U'$, assuming we are given $d_G(u, p_i)$ and $d_G(u, p_j)$. This shows the lemma when $D < \infty$.

Now, assume that $D = \infty$ and let u be a vertex in U'. Then for any $v \in R$, if $d_R(p_i, v) < \infty$ then v can be assigned colour c_i in an $\{i, j\}$-colouring of R w.r.t. u. And if $d_R(p_j, v) < \infty$ then v can be assigned colour c_j in an $\{i, j\}$-colouring of R w.r.t. u. Finally, if $d_R(p_i, v) = d_R(p_j, v) = \infty$ then v can be assigned either of the two colours c_i and c_j in an $\{i, j\}$-colouring of R w.r.t. u. This shows the lemma when $D = \infty$. □

We associate with T_2-entry $\overline{v}$ a level 3-table T_3. This table represents the set $\{0, \ldots, n_R\}^m$, where m is the number of sets in Lemma 8. Lemma 9 and the above observations show that if the vertices in a subset of U are mapped to the same T_3-entry then we can associate a colouring with this entry which is an $\mathcal{I}_w$-colouring of R w.r.t. each vertex in that subset.

We enumerate all entries of level 3-tables associated with vertex w of main tree $\mathcal{T}$ with integers $1, \ldots, m_w$, where m_w is the total number of entries. In the main algorithm, when a query vertex $u \in U$ is mapped to level 3-table entry with integer k then k is the integer returned. By the above, this value defines an $\mathcal{I}_w$-colouring of R w.r.t. u.

3.1.5 Construction Time

Let us bound the time for the preprocessing step and the size of the data structure obtained. We will assume that SSSP distances in G for each boundary vertex have been precomputed, and that for each region R in the r-division of G and for each cycle $C \in \mathcal{C}_R$, SSSP distances in $U_{R,C}$ have been precomputed for each boundary vertex of R belonging to C. The latter allows us to compute the length of all abstract edges in $O(r)$ time. All these SSSP distances can be computed in a total of $O(n^2/\sqrt{r})$ using the linear time SSSP algorithm of [9].

From the description and analysis of the preprocessing step, it is easy to see that it has running time at most a factor polynomial in r larger than the number of level 3-table entries, given the above precomputations. We will show that the main tree $\mathcal{T}$ and its associated data structures contain a total of $O(r^{O(\sqrt{r}\log r)})$ level 3-table entries. This will imply that the total running time of the preprocessing step is $O(r^{O(\sqrt{r}\log r)})$ in addition to the $O(n^2/\sqrt{r})$ time above.

Let w be a vertex of $\mathcal{T}$. We prove by induction on the height ≥ 0 of subtree $\mathcal{T}_w$ that the total number of level 3-table entries associated with w is at most $r^{c_1|\mathcal{I}_w|\log(|\mathcal{I}_w|)}$ for some constant c_1. Since $\mathcal{T}$ has $O(\sqrt{r})$ vertices, this

will show our claim.

If the height is zero then $\mathcal{T}_w$ is a leaf. Since a leaf has no associated data structure, our claim trivially holds in this case.

Now, suppose the height is at least one and that the induction hypothesis holds for smaller heights. Since the level 1-table T_w associated with w has a row resp. column for each level 3-table entry associated with $lc(w)$ resp. $rc(w)$, it follows from the induction hypothesis that T_w has at most $r^{c_1\lceil|\mathcal{I}_w|/2\rceil\log(\lceil|\mathcal{I}_w|/2\rceil)}$ rows and at most $r^{c_1(|\mathcal{I}_w|-\lceil|\mathcal{I}_w|/2\rceil)\log(|\mathcal{I}_w|-\lceil|\mathcal{I}_w|/2\rceil)}$ columns.

Consider some entry of T_w. The associated level-2 table has at most $r^{c_2|\mathcal{I}_w|}$ entries for some constant c_2. The number of level 3-table entries associated with each entry of that level-2 table is at most $r^{c_3|\mathcal{I}_w|}$ for some constant c_3. Hence, the total number of level 3-table entries associated with a single entry of T_w is at most $r^{c_4|\mathcal{I}_w|}$ where $c_4 = c_2c_3$.

Since the height is at least one, we have $|\mathcal{I}_w| > 1$, implying that $\lceil|\mathcal{I}_w|/2\rceil < \frac{3}{4}|\mathcal{I}_w|$. From this and the above, it follows that the total number of level 3-table entries associated with w is at most

$$\begin{aligned} r^{c_1(\lceil|\mathcal{I}_w|/2\rceil+|\mathcal{I}_w|-\lceil|\mathcal{I}_w|/2\rceil)\log(\lceil|\mathcal{I}_w|/2\rceil)+c_4|\mathcal{I}_w|} &= r^{c_1|\mathcal{I}_w|\log(\lceil|\mathcal{I}_w|/2\rceil)+c_4|\mathcal{I}_w|} \\ &< r^{c_1|\mathcal{I}_w|\log(\frac{3}{4}|\mathcal{I}_w|)+c_4|\mathcal{I}_w|} \\ &= r^{c_1|\mathcal{I}_w|\log(|\mathcal{I}_w|)+c_1\log(\frac{3}{4})|\mathcal{I}_w|+c_4|\mathcal{I}_w|} \end{aligned}$$

Thus, if we choose c_1 sufficiently large, i.e. such that $c_1\log(\frac{3}{4}) \leq -c_4$ then the total number of level 3-table entries associated with w is at most $r^{c_1|\mathcal{I}_w|\log(|\mathcal{I}_w|)}$, as requested. This gives the following result.

Theorem 1. *The number of level 3-table entries associated with main tree $\mathcal{T}$ and the total time spent in the preprocessing step are $O(r^{O(\sqrt{r}\log r)})$, assuming SSSP distances for boundary vertices are given.*

3.2 Main Algorithm

We are now ready to describe the main algorithm. We will show how the precomputed main tree $\mathcal{T}$ and its associated data structures can be used to efficiently find an integer representing a u-colouring of R for each vertex $u \in U$.

So let u be one such vertex. We start at the root r of $\mathcal{T}$ and recursively find two integers, i_{lc} and i_{rc}, representing, respectively, an $\mathcal{I}_{lc(r)}$-colouring and

an $\mathcal{I}_{rc(r)}$-colouring of R w.r.t. u. Using these integers as indices, we obtain the tree set $\mathcal{TS}$ associated with entry (i_{lc}, i_{rc}) in level 1-table T_r.

We perform a u-traversal in $\mathcal{TS}$ which gives us a binary tree and we map this tree to a vector defining an entry in the level 2-table associated with $\mathcal{TS}$. Let T_3 be the level 3-table associated with this entry.

Finally, we use the algorithm implicit in Lemma 9 to obtain the correct entry of T_3. The integer associated with this entry is then returned as that representing a u-colouring of R.

The following theorem shows how the main algorithm can efficiently compute the integer representing a u-colouring of R for any $u \in U$.

Lemma 10. *Given main tree $\mathcal{T}$ with associated data structures and given SSSP distances in G for each of the boundary vertices $p_1, \ldots, p_t$ of R in C, the main algorithm obtains the integer representing a u-colouring of R in time $O(t \log^2 r)$ for any $u \in U$ with preprocessing time polynomial in r.*

Proof. Let $u \in U$ be given. Let w be any non-leaf vertex of main tree $\mathcal{T}$ and assume that we are given the two integers i_{lc} and i_{rc} representing, respectively, an $\mathcal{I}_{lc(w)}$-colouring and an $\mathcal{I}_{rc(w)}$-colouring of R w.r.t. u. We will show that we can obtain the integer representing an $\mathcal{I}_w$-colouring of R w.r.t. u in $O(|\mathcal{I}_w| \log r)$ time. This will imply the theorem since $\mathcal{T}$ has height $O(\log r)$, since the sum of the sizes of index sets associated with vertices of the same depth in $\mathcal{T}$ is $O(|\mathcal{I}_w|)$, and since $|\mathcal{I}_w| = t$ when w is the root of $\mathcal{T}$.

We can obtain the tree set $\mathcal{TS}$ associated with entry (i_{lc}, i_{rc}) of the level 1-table of w in constant time. It suffices to show that the binary tree of $\mathcal{TS}$ traversed by u can be found in $O(|\mathcal{I}_w| \log r)$ time. For suppose this tree is given. Then a depth-first traversal of it gives the vector mapping the tree to an entry of the associated level 2-table in $O(|\mathcal{I}_w|)$ time by Lemma 6. And given this entry, we can find the entry in the appropriate level 3-table in $O(|\mathcal{I}_w| \log r)$ time with preprocessing time polynomial in r by Lemma 8 and 9.

To show that the tree of $\mathcal{TS}$ traversed by u can be found in $O(|\mathcal{I}_w| \log r)$ time we will show that the appropriate entry in one of the two arrays $\mathcal{A}_i$ and $\mathcal{A}_j$ at the top-level of a tree set $\mathcal{TS}'$ can be found in $O(\log r)$ time. This will show our claim since the tree traversed by u has $O(|\mathcal{I}_w|)$ vertices by Lemma 6.

Let subset A, index sets $\mathcal{I} \subseteq \mathcal{I}_{lc(w)}$, $\mathcal{I}' \subseteq \mathcal{I}_{rc(w)}$, colourings $c : A \to \mathcal{I}$ and $c' : A \to \mathcal{I}'$, vertex v and paths P_i and P_j be associated with $\mathcal{TS}'$. Deciding which of the two arrays $\mathcal{A}_i$ and $\mathcal{A}_j$ contains the entry we are looking

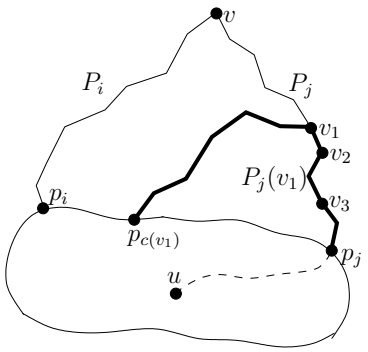

Figure 7: The situation in the proof of Lemma 10.

for can be done in constant time since this involves determining whether $d_G(u, p_i) + d_R(p_i, v) \leq d_G(u, p_j) + d_R(p_j, v)$ and since SSSP distances have been precomputed for each boundary vertex.

Suppose w.l.o.g. that $d_G(u, p_i) + d_R(p_i, v) \leq d_G(u, p_j) + d_R(p_j, v)$, i.e. that the entry belongs to $\mathcal{A}_j$. Let us use the same notation as in the construction of $\mathcal{TS}'$. Then we need to find in P_j a split vertex of A w.r.t. u and colourings c and c'.

We observe that the redefinition of c' in the construction of $\mathcal{TS}'$ ensures that each vertex of P_j has colour j in the colouring c'. Hence, our claim will follow if we can find in $O(\log r)$ time a vertex $v_1 \in P_j$ such that

1. $d_G(u, p_{c(v_1)}) + d_R(p_{c(v_1)}, v_1) \leq d_G(u, p_j) + d_R(p_j, v_1)$ and
2. $d_G(u, p_j) + d_R(p_j, v_2) < d_G(u, p_k) + d_R(p_k, v_2)$ for any vertex $v_2 \in P_j(v_1) \setminus \{v_1\}$ and any $k \in \mathcal{I}$,

where $P_j(v_1)$ is the subpath of P_j from p_j to v_1.

The invariant in the construction of $\mathcal{TS}'$ ensures that every vertex of P_j belongs to A. Since c is an $\mathcal{I}$-colouring of A w.r.t. u, the second condition above can thus be restated as the simpler condition that $d_G(u, p_j) + d_R(p_j, v_2) < d_G(u, p_{c(v_2)}) + d_R(p_{c(v_2)}, v_2)$ for any vertex $v_2 \in P_j(v_1) \setminus \{v_1\}$. We will show that we can apply binary search in $\mathcal{A}_j$ to find a vertex v_1 satisfying this condition and the first condition above.

So consider some v_1-entry e of $\mathcal{A}_j$ and let (i_1, j) be the colour pair of v_1, see Figure 7. If the first condition is not satisfied for v_1 then it is not satisfied for any vertex in $P_j(v_1)$. So in this case, we can disregard e and all entries to one side of e in $\mathcal{A}_j$ when looking for a split vertex.

Now, assume that the first condition does hold for v_1. Let v_2 be the last vertex of $P_j(v_1) \setminus \{v_1\}$ when walking from p_j (we assume that such a vertex exists since otherwise, v_1 must be a split vertex). Let (i_2, j) be the colour pair of v_2.

If $d_G(u, p_{i_2}) + d_R(p_{i_2}, v_2) \leq d_G(u, p_j) + d_R(p_j, v_2)$ then a split vertex must belong to $P_j(v_1) \setminus \{v_1\}$ and again we can disregard e and all entries to one side of e in $\mathcal{A}_j$ when looking for a split vertex.

Finally, suppose the first condition is satisfied and $d_G(u, p_{i_2}) + d_R(p_{i_2}, v_2) > d_G(u, p_j) + d_R(p_j, v_2)$. We claim that in this case, v_1 is a split vertex.

To see this, we need to show that $d_G(u, p_j) + d_R(p_j, v_3) < d_G(u, p_{i_3}) + d_R(p_{i_3}, v_3)$ for any $v_3 \in P_j(v_1) \setminus \{v_1\}$, where (i_3, j) is the colour pair of v_3. Suppose for the sake of contradiction that $d_G(u, p_j) + d_R(p_j, v_3) \geq d_G(u, p_{i_3}) + d_R(p_{i_3}, v_3)$ for one such vertex. Then

$$\begin{aligned} d_G(u, p_j) + d_R(p_j, v_2) &= d_G(u, p_j) + d_R(p_j, v_3) + d_R(v_3, v_2) \\ &\geq d_G(u, p_{i_3}) + d_R(p_{i_3}, v_3) + d_R(v_3, v_2) \\ &\geq d_G(u, p_{i_3}) + d_R(p_{i_3}, v_2) \\ &\geq d_G(u, p_{i_2}) + d_R(p_{i_2}, v_2), \end{aligned}$$

contradicting the inequality $d_G(u, p_{i_2}) + d_R(p_{i_2}, v_2) > d_G(u, p_j) + d_R(p_j, v_2)$.

We have shown that binary search in $\mathcal{A}_j$ can be applied to find a split vertex. Since SSSP distances for each boundary vertex have been precomputed and since $\mathcal{A}_j$ has $O(r)$ entries, it follows that it takes $O(\log r)$ time to find a split vertex. From the discussion above, this suffices to prove the lemma. □

We are now ready for our first main result. In the following theorem, we state our generic algorithm and bound its running time.

Theorem 2. *With the above definitions, suppose $r = (c \log n/(\log\log n)^2)^2$ where c is a constant. If c is sufficiently small then there is a constant $\epsilon < 1$, an integer $N = O(n^\epsilon)$, a map $f : U \to \{1, \ldots, N\}$, and $\mathcal{I}_t$-colourings $c_1, \ldots, c_N$ of R where $c_{f(u)}$ is a u-colouring of R for all $u \in U$. The integers $f(u)$ for $u \in U$ and colourings $c_1, \ldots, c_N$ can be computed in a total of $O(n^\epsilon + |U| \log n)$ time with $O(n^2(\log\log n)^2/\log n)$ preprocessing time (independent of U and R).*

Proof. SSSP distances may be precomputed in $O(n^2/\sqrt{r})$ time.

In the main algorithm, vertices of U are mapped to integers in the range $\{1, \ldots, N\}$, where N is the total number of level 3-table entries. By Theorem 1 and Lemma 10 this takes a total of

$$O(r^{c'\sqrt{r}\log r} + |U|\sqrt{r}\log^2 r)$$

time for some constant c'. So to show one part of the theorem, that integers $f(u)$ for $u \in U$ can be computed in a total of $O(n^\epsilon + |U|\log n)$ time for some constant $\epsilon < 1$, we need to pick constant c such that $\sqrt{r}\log^2 r = O(\log n)$, $r^{c'\sqrt{r}\log n} = O(n^\epsilon)$, and $n^2/\sqrt{r} = O(n^2(\log\log n)^2/\log n)$. This also shows $N = O(n^\epsilon)$.

For sufficiently large n,

$$\begin{aligned}
r^{c'\sqrt{r}\log r} &= (c\log n/(\log\log n)^2)^{2c'(c\log n/(\log\log n)^2)\log((c\log n/(\log\log n)^2)^2)} \\
&\leq (c\log n)^{(4c'c\log n/(\log\log n)^2)\log(c\log n)} \\
&\leq (c\log n)^{(8c'c\log n/(\log\log n)^2)\log\log n} \\
&\leq (\log n)^{16c'c\log n/\log\log n} \\
&= 2^{16c'c\log n} \\
&= n^{16c'c}.
\end{aligned}$$

By setting $c < 1/(16c')$, we have $r^{c'\sqrt{r}\log n} = O(n^\epsilon)$ with $\epsilon < 1$. Also,

$$\sqrt{r}\log^2 r = O((\log n/(\log\log n)^2)(\log\log n)^2) = O(\log n).$$

And finally, $n^2/\sqrt{r} = O(n^2(\log\log n)^2/\log n)$, as requested.

To bound the time to compute the colourings, we observe that each colouring can be obtained in time polynomial in r. Thus, the total time to compute all colourings is $O(Nr^{O(1)}) = O(r^{O(\sqrt{r}\log r)})$. This is $O(n^\epsilon)$ for c sufficiently small. □

4 Applications of the Generic Algorithm

In this section, we show how the generic algorithm gives us, almost for free, subquadratic time algorithms for computing the (weighted) Wiener index, the diameter, and the stretch factor of a planar graph with non-negative edge-weights.

We start with the problem of computing the Wiener index of a planar graph $G = (V, E)$ with non-negative edge-weights. For $V_1, V_2 \subseteq V$, define $\sum(V_1, V_2) = \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} d_G(v_1, v_2)$. We extend this definition to subgraphs of G by summing over their vertex sets. The *Wiener index* of G is defined as $\frac{1}{2}\sum(V, V)$.

Theorem 3. *The Wiener index of an n-vertex planar graph with non-negative edge weights can be computed in $O(n^2(\log\log n)^4/\log n)$ time.*

Proof. Let G be an n-vertex planar graph with non-negative edge weights. We start by computing an r-division of G in $O(n \log n)$ time with $r = (c \log n/(\log\log n)^2)^2$ for constant c satisfying Theorem 2. We also precompute SSSP distances in G for each boundary vertex. As we saw earlier, this can be done in $O(n^2(\log\log n)^2/\log n)$ time.

By Lemma 2,

$$\sum G = \frac{1}{2} \sum_{R \in \mathcal{R}} \left(\sum(R, R) + \sum_{C \in \mathcal{C}_R} \sum(U_{R,C}, R) \right).$$

Let $R \in \mathcal{R}$ be given. We will show how to compute $\sum(R, R)$ and $\sum_{C \in \mathcal{C}_R} \sum(U_{R,C}, R)$ in $O(|\mathcal{C}_R| n^\epsilon + n \log n)$ time for some constant $\epsilon < 1$. Since each cycle occurs in at most two regions and since there are $O(n/r)$ regions it will follow from this that $\sum G$ can be computed in time

$$\begin{aligned} O\left(\frac{n}{r} n^\epsilon + \frac{n}{r} n \log n\right) &= O\left(\frac{n^{1+\epsilon}}{r} + \frac{n^2 \log n}{(\log n/(\log\log n)^2)^2}\right) \\ &= O(n^2(\log\log n)^4/\log n). \end{aligned}$$

To compute $\sum(R, R)$, let R' be the graph obtained from R by adding an edge between each pair of boundary vertices of R. The weight of each edge is equal to the distance in G between the end vertices of the edge. We compute APSP distances in R' and obtain $\sum(R, R)$ by adding up all these distances. The time it takes to add edges and compute their weights is $O(r)$ time, given the precomputed SSSP distances. It then takes $O(r^3)$ time to compute APSP distances by using an algorithm like Floyd-Warshall. Thus, $\sum(R, R)$ can be computed in time polylogarithmic in n.

What remains is to show that $\sum_{C \in \mathcal{C}_R} \sum(U_{R,C}, R)$ can be computed in $O(|\mathcal{C}_R| n^\epsilon + n \log n)$ time for some constant $\epsilon < 1$ with $O(n^2(\log\log n)^2/\log n)$

preprocessing time. So let $C \in \mathcal{C}_R$ be given and let $p_1, \ldots, p_t$ be the boundary vertices of R belonging to C.

By Theorem 2, there is a constant $\epsilon' < 1$, an integer $N = O(n^{\epsilon'})$, a map $f : U_{R,C} \to \{1, \ldots, N\}$, and $\mathcal{I}_t$-colourings $c_1, \ldots, c_N$ of R such that $c_{f(u)}$ is a u-colouring of R for all $u \in U_{R,C}$. The integers $f(u)$ for $u \in U_{R,C}$ and colourings $c_1, \ldots, c_N$ can be computed in a total of $O(n^{\epsilon'} + |U_{R,C}| \log n)$ time with $O(n^2(\log\log n)^2/\log n)$ preprocessing time.

Let $M \in \{1, \ldots, N\}$. For the colouring c_M of R corresponding to M, we compute $\sum(\{p_i\}, V_{i,M})$ for $i = 1, \ldots, t$, where $V_{i,M}$ is the set of vertices of R with colour i. We also compute the number $|V_{i,M}|$ of vertices in $V_{i,M}$. This can clearly be done in time polynomial in r which is poly-logarithmic in n. Over all i and M, this is $\tilde{O}(n^{\epsilon'}) = O(n^{\epsilon})$ time for some constant $\epsilon < 1$.

Now, for a vertex $u \in U_{R,C}$, let $M_u = f(u)$. Then

$$\sum(\{u\}, R) = \sum_{i=1}^{t} d_G(u, p_i)|V_{i,M_u}| + \sum(\{p_i\}, V_{i,M_u}). \tag{2}$$

Given the above precomputations, it follows that $\sum(\{u\}, R)$ can be computed in $O(t)$ time. Hence, $\sum(U_{R,C}, R)$ can be computed in $O(n^{\epsilon} + |U_{R,C}|t) = O(n^{\epsilon} + |U_{R,C}|\sqrt{r})$ time with $O(n^2(\log\log n)^2/\log n)$ preprocessing time.

Adding this up over all $C \in \mathcal{C}_R$ and using the fact that $\sum_{C \in \mathcal{C}_R} |U_{R,C}| \leq n$, it follows that $\sum_{C \in \mathcal{C}_R} \sum(U_{R,C}, R)$ can be computed in

$$O(|\mathcal{C}_R|n^{\epsilon} + n\sqrt{r}) = O(|\mathcal{C}_R|n^{\epsilon} + n\log n/(\log\log n)^2)$$

time in addition to the $O(|\mathcal{C}_R|n^{\epsilon'} + n\log n)$ time spent in Theorem 2 and the $O(n^2(\log\log n)^2/\log n)$ preprocessing time. □

Next, the diameter of a planar graph.

Theorem 4. *The diameter of an n-vertex planar graph with non-negative edge weights can be computed in $O(n^2(\log\log n)^4/\log n)$ time.*

Proof. The proof is similar to that of Theorem 3. The only essential difference is that we compute $\max_{v \in V_{i,M_u}} d_G(p_i, v)$ instead of $\sum(\{p_i\}, V_{i,M_u})$ and use the identity

$$\max_{v \in R} d_G(u, v) = \max\left\{ d_G(u, p_i) + \max_{v \in V_{i,M_u}} d_G(p_i, v) | i = 1, \ldots, t \right\}$$

instead of (2). □

Finally, the stretch factor of a plane geometric graph $G = (V, E)$. For subsets $V_1, V_2 \subseteq V$, define

$$\delta_G(V_1, V_2) = \max_{v_1 \in V_1, v_2 \in V_2, v_1 \neq v_2} \frac{d_G(v_1, v_2)}{\|v_1 v_2\|_2}.$$

We extend this definition to subgraphs of G by taking the maximum over their vertex sets. The *stretch factor* of G is defined as $\delta_G(V, V)$.

Theorem 5. *The stretch factor of an n-vertex plane geometric graph can be computed in $O(n^2(\log\log n)^4/\log n)$ expected time.*

Proof. Let G be an n-vertex plane geometric graph. We first compute an r-division of a triangulation of G with $r = (c\log n/(\log\log n)^2)^2$ for constant c satisfying Theorem 3 and compute SSSP distances for all boundary vertices.

Let R be a region in this r-division. We will show how to compute $\delta_G(U_{R,C}, R)$ for all $C \in \mathcal{C}_R$ using a total of $O(|\mathcal{C}_R|n^\epsilon + n\log n)$ expected time for some constant $\epsilon < 1$ with $O(n^2(\log\log n)^2/\log n)$ preprocessing time. As in the proof of Theorem 3, it will follow from this that the stretch factor of G can be computed in $O(n^2(\log\log n)^4/\log n)$ expected time.

Let $C \in \mathcal{C}_R$ and let $p_1, \ldots, p_t$ be the boundary vertices of R belonging to C. By Theorem 2, there is a constant $\epsilon' < 1$, an integer $N = O(n^{\epsilon'})$, a map $f : U_{R,C} \to \{1, \ldots, N\}$, and $\mathcal{I}_t$-colourings $c_1, \ldots, c_N$ of R such that $c_{f(u)}$ is a u-colouring of R for all $u \in U_{R,C}$. The integers $f(u)$ for $u \in U_{R,C}$ and colourings $c_1, \ldots, c_N$ can be computed in a total of $O(n^{\epsilon'} + |U_{R,C}|\log n)$ time with $O(n^2(\log\log n)^2/\log n)$ preprocessing time.

For $M = 1, \ldots, N$, define the *group* U_M as the set of vertices $u \in U_{R,C}$ such that $f(u) = M$. All vertices in the same group induce identical colourings of R.

Consider one such group U_M. For $i = 1, \ldots, t$, lift each vertex $v \in V_{i,M}$ to height $d_G(p_i, v)$ on the z-axis, where $V_{i,M}$ is the set of vertices of R with colour i in the colouring c_M associated with group U_M. Furthermore, lower each vertex u of U_M to height $-d_G(u, p_i)$. Each lifting/lowering of a vertex takes constant time given the precomputed SSSP distances. The sets $V_{i,M}$ over all i and M can be computed in $\tilde{O}(n^{\epsilon'}) = O(n^\epsilon)$ time for some $\epsilon > 0$.

Observe that the height difference between any lowered vertex u and any lifted vertex v is equal to $d_G(u, v)$. Now, arbitrarily divide U_M into $O(|U_M|/\sqrt{r})$ subsets each containing $O(\sqrt{r})$ vertices. Applying the algorithm

of [1] to the (lowered) vertices in each of these subsets and to the (lifted) vertices in $V_{i,M}$ gives $\delta_G(U_M, V_{i,M})$ in expected time

$$O\left(\frac{|U_M|}{\sqrt{r}}(\sqrt{r}+|V_{i,M}|)\log(\sqrt{r}+|V_{i,M}|)\right) = O\left(|U_M|\left(1+\frac{|V_{i,M}|}{\sqrt{r}}\right)\log r\right).$$

Summing over all i, we see that $\delta_G(U_M, R)$ can be found in expected time

$$O(|U_M|(t+r/\sqrt{r})\log r) = O(|U_M|\sqrt{r}\log r).$$

Hence, $\delta_G(U_{R,C}, R)$ can be found in $O(|U_{R,C}|\sqrt{r}\log r)$ expected time. Over all $C \in \mathcal{C}_R$, this is $O(n\sqrt{r}\log r) = O(n\log n/\log\log n)$. This shows the theorem. □

Using parametric search as in [1], we can obtain $\delta_G(U_{R,C}, R)$ in *worst-case* time $O(|U_{R,C}|\sqrt{r}(\log r)^{O(1)})$ in the proof of Theorem 5. Over all $C \in \mathcal{C}_R$, this is $O(n\log n(\log\log n)^{O(1)})$, implying that the stretch factor of G can be computed in $O(n^2(\log\log n)^{O(1)}/\log n)$ worst-case time.

5 Unweighted Graphs

In [14], we showed that the Wiener index of an n-vertex planar unweighted graph can be found in $O(n^2\log\log n/\log n)$ time using $O(n)$ space. We only needed planarity to obtain an r-division (as defined in that paper) and to find SSSP path distances in linear time. As noted in [9], both of these results hold for any subgraph-closed $\sqrt{n}$-separable graph (see definition in [9]). And by applying the same idea as in the proof of Theorem 4, it follows that both the Wiener index and the diameter of such a graph can be found in $O(n^2\log\log n/\log n)$ time and $O(n)$ space.

Theorem 6. *The Wiener index and diameter of an unweighted n-vertex subgraph-closed $\sqrt{n}$-separable graph can be found in $O(n^2\log\log n/\log n)$ time with $O(n)$ space.*

6 Concluding Remarks

We showed how to compute the Wiener index and the diameter of an n-vertex planar graph with non-negative edge-weights in $O(n^2(\log\log n)^4/\log n)$ worst-case time and the stretch factor of an n-vertex plane geometric graph in

$O(n^2(\log\log n)^4/\log n)$ expected time. Previously, it was open whether any of these three problems could be solved in subquadratic time. We also showed that the Wiener index and diameter of an unweighted n-vertex subgraph-closed $\sqrt{n}$-separable graph can be found in $O(n^2 \log\log n/\log n)$ time with $O(n)$ space.

Our results for weighted graphs are obtained by applying the same generic algorithm. We hope this algorithm may yield faster algorithms for other planar graph problems involving, in some way, shortest path distances between all or some pairs of vertices.

We pose the following questions: is there a constant $c < 2$ such that any of the above problems can be solved in $O(n^c)$ time? Are there subquadratic time algorithms for computing the Wiener index, diameter, and stretch factor of an even larger class of graphs, such as the class of subgraph-closed $\sqrt{n}$-separable graphs with non-negative edge weights? What about lower bounds on running time?

References

[1] P. K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, M. Sharir, and M. Soss. Computing the Detour and Spanning Ratio of Paths, Trees and Cycles in 2D and 3D. Discrete and Computational Geometry, 39 (1): 17–37 (2008).

[2] S. Cabello. Many distances in planar graphs. In Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1213–1220, New York, NY, USA, 2006. ACM Press.

[3] S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. Manuscript, Berlin, 2007.

[4] V. Chepoi and S. Klavžar. The Wiener index and the Szeged index of benzenoid systems in linear time. J. Chem. Inf. Comput. Sci., 37:752–755, 1997.

[5] F. R. K. Chung. Diameters of Graphs: Old Problems and New Results. Congressus Numerantium, 60:295–317, 1987.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. McGraw-Hill, 2nd ed., 2001.

[7] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. SIAM J. Comput., 16 (1987), pp. 1004–1022.

[8] Y. Han. An $O(n^3(\log\log n/\log n)^{5/4})$ Time Algorithm for All Pairs Shortest Paths. In Proc. of ESA, Springer-Verlag LNCS 4168:411–417, 2006.

[9] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster Shortest-Path Algorithms for Planar Graphs. Journal of Computer and System Sciences volume 55, issue 1, August 1997, pages 3–23.

[10] R. J. Lipton and R. E. Tarjan. A Separator Theorem for Planar Graphs. STAN-CS-77-627, October 1977.

[11] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. J. Comput. Syst. Sci., 32:265–279, 1986.

[12] B. Mohar and T. Pisanski. How to compute the Wiener index of a graph. J. Math. Chem., pages 267–277, 1988.

[13] H. Wiener. Structural determination of paraffin boiling points. J. Amer. Chem. Sot., 69:17–20, 1947.

[14] C. Wulff-Nilsen. Sum of All-Pairs Shortest Path Distances in a Planar Graph in Subquadratic Time. Technical Report 08/11, Department of Computer Science, University of Copenhagen, 2008.

[15] B. Zmazek and J. Žerovnik. Computing the weighted Wiener and Szeged number on weighted cactus graphs in linear time. Croatica Chemica Acta, 76:137–143, 2003.